

Evaluation of Top-k OLAP Queries Using Aggregate R-trees



Nikos Mamoulis (HKU)

Spiridon Bakiras (HKUST)

Panos Kalnis (NUS)

Background

- ❑ On-line Analytical Processing (OLAP) refers to the set of operations that are applied on a Data Warehouse to assist analysis and decision support.
- ❑ Some measures (e.g., sales) are summarized with respect to some interesting dimensions (e.g., products, stores, time, etc.), representing business perspectives.
- ❑ E.g., “retrieve the total sales per month, product-color and store location”.

Background

- **Fact table:** stores measures and values of all dimensions at the most refined abstraction level.
- **Dimensional tables:** store information about the multi-level hierarchies of each dimension.
- Some of the dimensions could be **spatial** (e.g., location). Hierarchies for spatial attributes may exist (e.g., exact location, district, city, county, state, country).

Background

- **OLAP queries:** summarize measures based on selected dimensions and hierarchies thereof.

- E.g.:

```
SELECT product-type, store-city, sum(quantity)
FROM Sales
GROUP BY product-type, store-city
```

Background

- ❑ **Evaluation of OLAP queries** can be performed by expensive joins on the base data, or can be assisted by **materialized views** which are pre-computed and maintained summaries of the warehouse at some abstraction level.
- ❑ Not all combinations of dimensions/hierarchies can be materialized; too expensive to store and maintain.
- ❑ **View selection:** materialize a set of views that minimize the cost of expected queries, based on space/time constraints.

Top-k OLAP Queries

- a **top-k OLAP query** selects the k groups with the highest aggregate values.

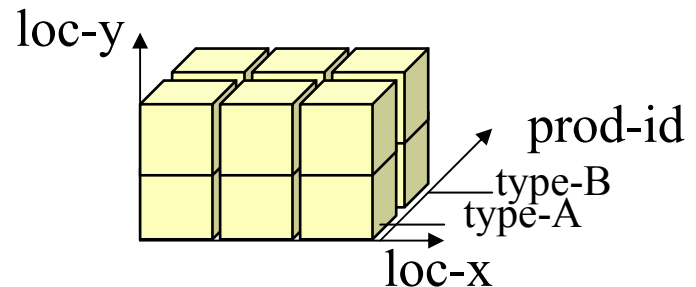
```
SELECT product-type, store-city, sum(quantity)
FROM Sales
GROUP BY product-type, store-city
ORDER BY sum(quantity)
STOP AFTER k
```

- related: **iceberg query**

```
SELECT product-type, store-city, sum(quantity)
FROM Sales
GROUP BY product-type, store-city
HAVING sum(quantity) > 1000
```

Problem formulation

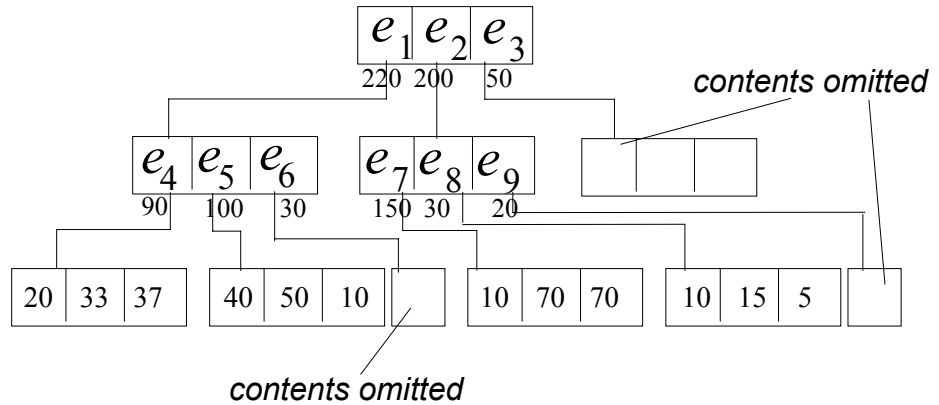
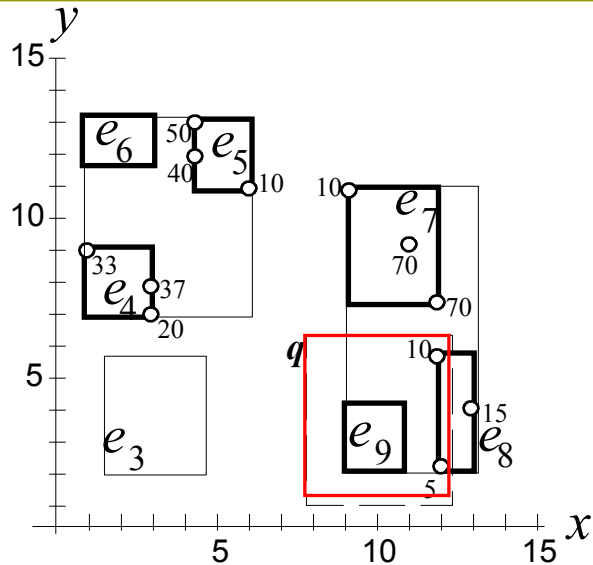
- $D = \{d_1, d_2, \dots, d_n\}$ a set of interesting dimensions (some dimensions could be spatial)
 - e.g., product, location, etc.
- The values of each dimension d_i are partitioned to a set R_i of ranges, either ad-hoc or based on some hierarchy level.
 - e.g., product-ids partitioned to product-types
 - e.g., spatial dimension is partitioned using a grid
- Retrieve the k multi-dimensional groups with the greatest values



Assumption

- We assume that the set of interesting dimensions is already indexed by an **aggregate R-tree (aR-tree)**
- Realistic, by view selection on the most-refined hierarchical level
 - We do not require hierarchical data summaries to be indexed.
- Objective:
 - Evaluate top-k OLAP queries (and iceberg queries) using the aR-tree.

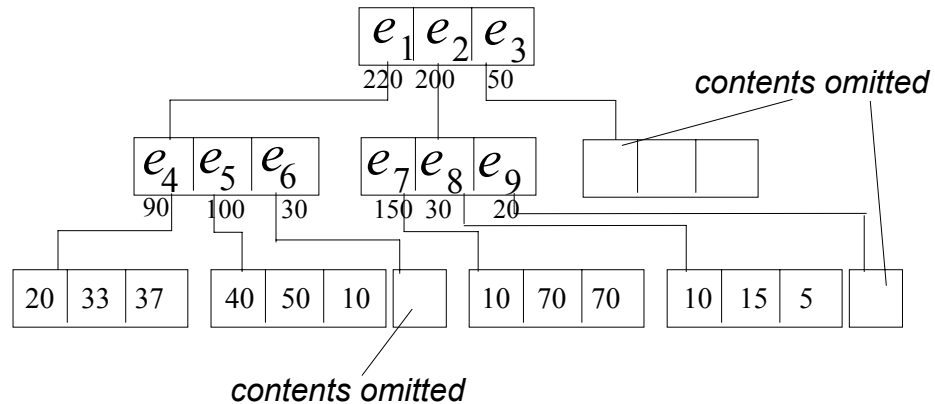
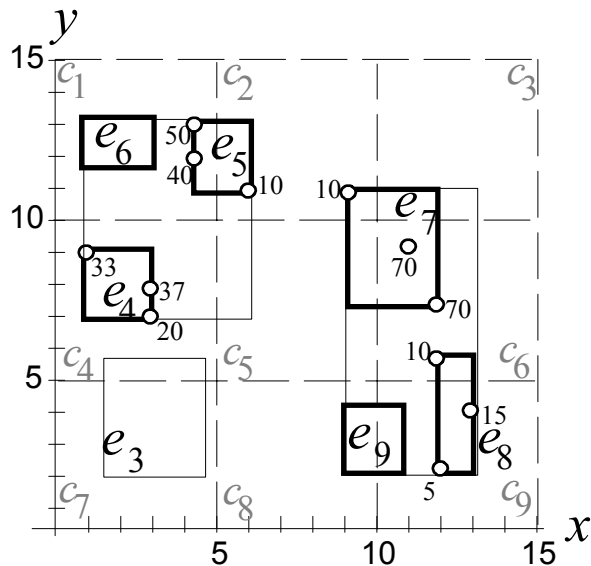
The aggregate R-tree



- Each entry is augmented with aggregate information about all objects indexed in the subtree pointed by it.
- Simple aggregate range queries:** retrieving the aggregate for a region that spatially covers an entry does not require accessing the corresponding node.

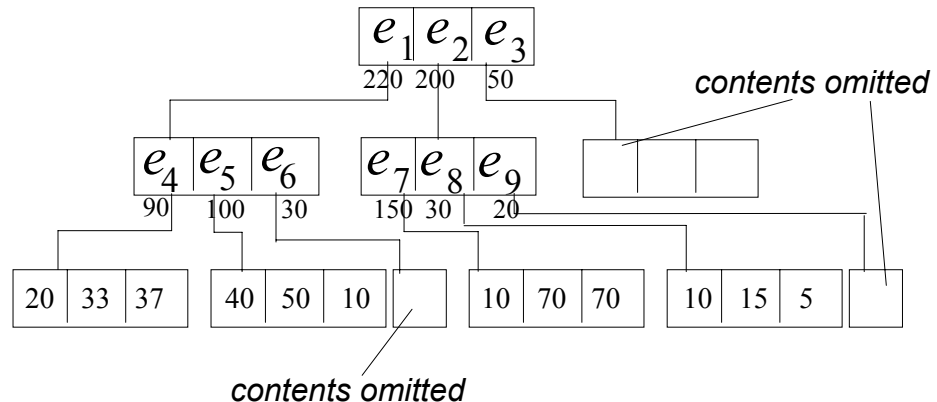
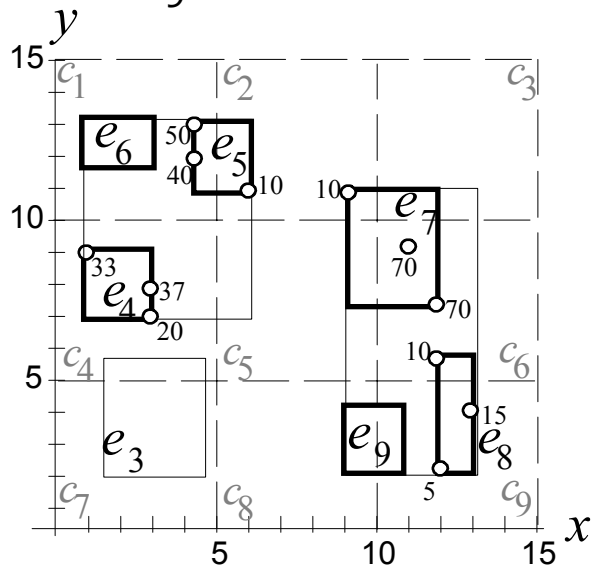
Using the aR-tree for top-k OLAP queries

- **Observation:** By browsing the tree partially, we can derive upper and lower bounds for the aggregate value of each cell:
 - E.g., $c_1.\text{agg} = 120$, $90 \leq c_4.\text{agg} \leq 140$
 - c_6 (with $c_6.\text{agg} = 150$) is the top-1 result



Using the aR-tree for top-k OLAP queries

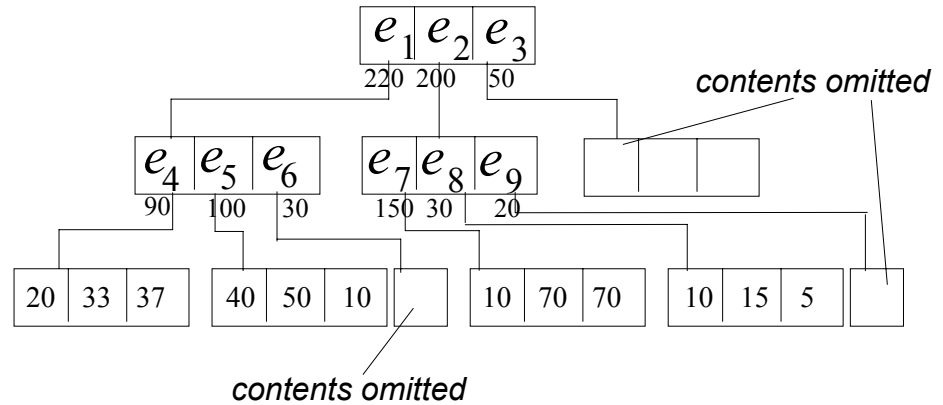
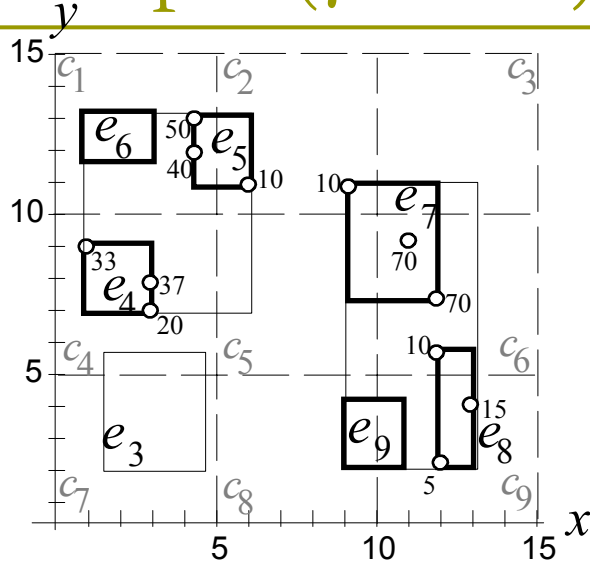
- Lemma:** Let t be the k -th largest lower bound of all cells. Let e_i be an aR-tree entry. If for all cells c that intersect e_i , $c.ub \leq t$, then the subtree pointed by e_i cannot contribute to any top-k result, and thus it can be pruned from search.
 - E.g., $t=c_6.agg=150$, e_9 intersects c_8, c_9 , $c_8.ub=20$, $c_9.ub=40$



Sketch of basic algorithm

- ❑ Assume (for now) that information about all cube cells (upper and lower aggregate bounds) can be maintained in memory.
- ❑ Initialize $c.lb=c.ub=0$, for all cells.
- ❑ Maintain a heap H with the non-visited entries yet. Initially H contains the root entries of the aR-tree. Update $c.lb$, $c.ub$ for all cells based on their intersection/containment of entries.
- ❑ Maintain a heap LB with the cells of top- k lower bounds. Let t be the lowest $c.lb$.
- ❑ Each entry e in H is prioritized based on
 - $e.ub = \max\{c.ub, \text{for all } c \text{ intersected by } e\}$
- ❑ While $\text{top}(H).ub > t$, remove top entry from H , visit the corresponding R-tree node and update upper/lower bounds and LB .

Example ($\gamma = \text{sum}$)



- visit root; $t=0$; $c_2.\text{ub}=420$; pick e_1 (greatest e.agg);
- visit $e_1.\text{ptr}$; $t=90=c_4.\text{lb}$; $c_2.\text{ub}=300$; pick e_2 ;
- visit $e_2.\text{ptr}$; $t=90=c_4.\text{lb}$; $c_2.\text{ub}=250$; pick e_7 ;
- visit $e_7.\text{ptr}$; $t=140=c_6.\text{lb}$; $c_6.\text{ub}=170$; pick e_6 ;
- visit $e_6.\text{ptr}$; $t=150=c_6.\text{lb}$; $c_4.\text{ub}=140 < t$; terminate

Reducing Memory Requirements

- ❑ The basic algorithm requires maintenance of lower/upper bounds for each cell. This might be infeasible in practice (huge number of groups).
- ❑ Optimizations
 - need not keep information about cells that are intersected by at most one entry
 - keep a single upper bound for all cells intersected by the same set of entries
 - need not keep information about cells that may not end up in the top-k result
 - maintain upper bounds only at tree entries (not at cells)

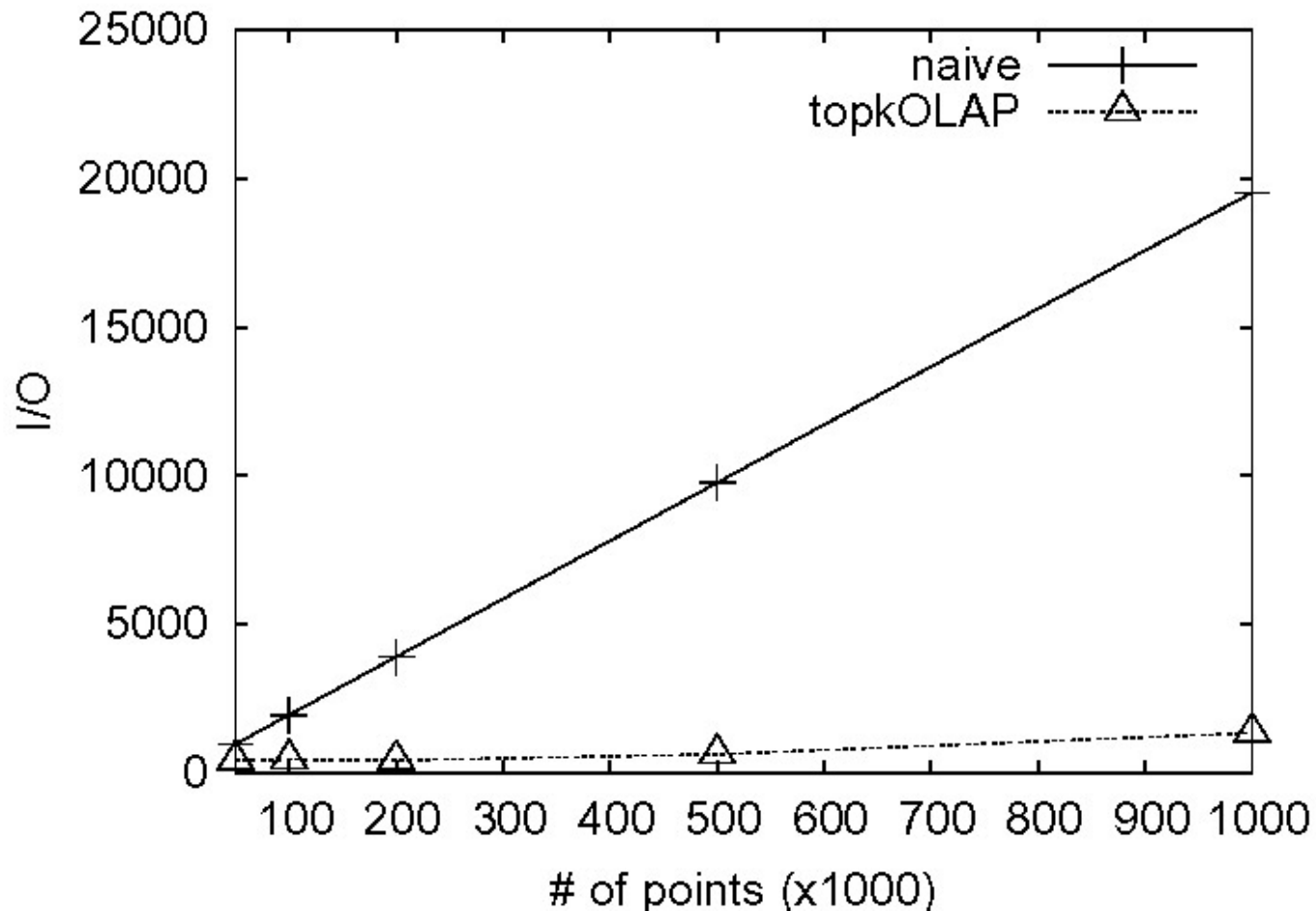
Extensions

- ❑ **Iceberg queries.** Similar algorithm; replace floating bound of k-th c.lb by constant t. No need of a priority queue; visit nodes in depth-first order.
- ❑ **Range-restricted top-k OLAP queries.** Use selection range to prune.
- ❑ **Non-orthocanonical partitions.** Use a bipartite graph that links tree entries to regions of partitions
- ❑ **Multiple measures and different aggregate functions.** Easy to extend assuming that the tree stores aggregates (e.g., sum, min, etc.)

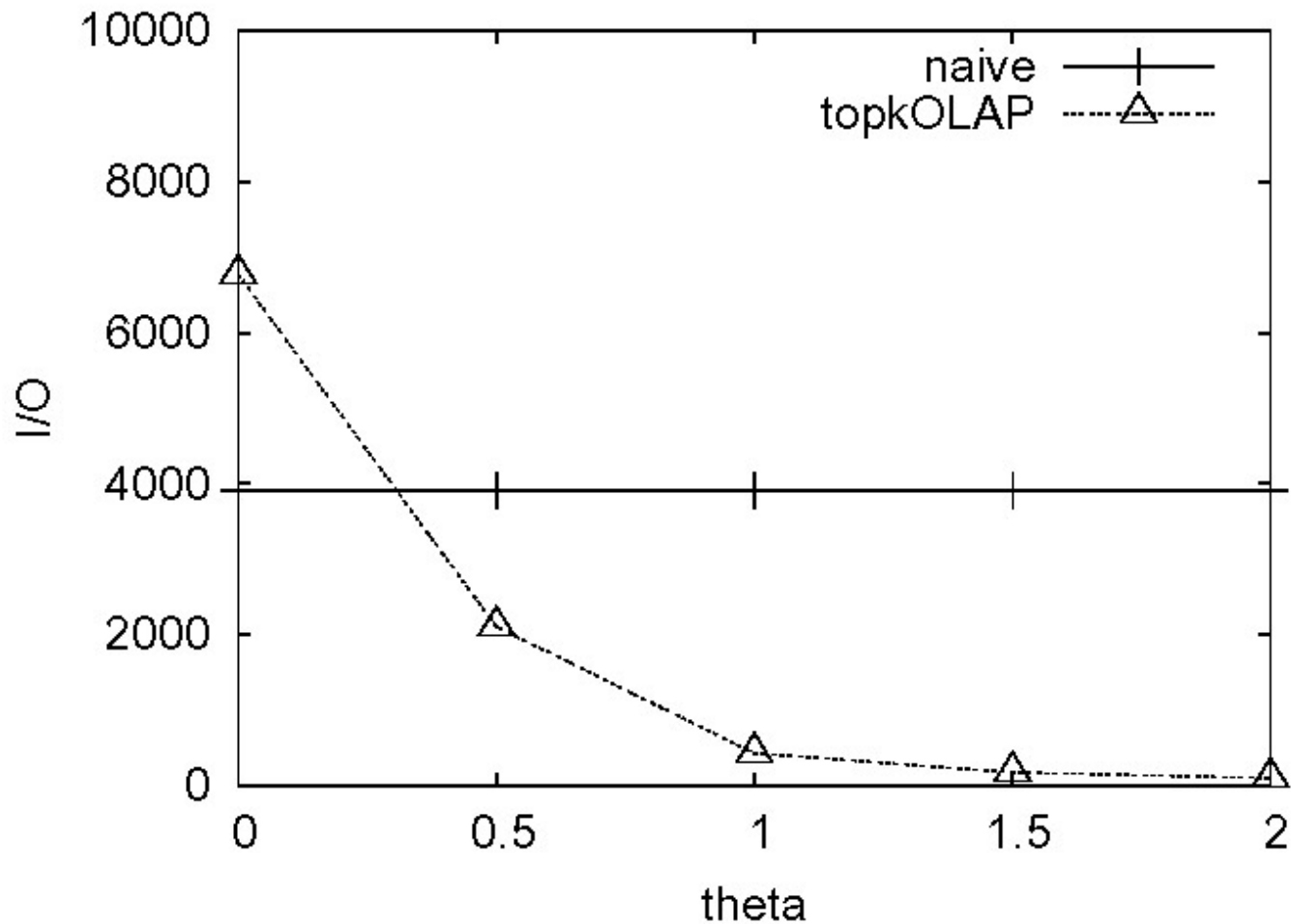
Experimental Settings

- **Synthetic data (uniform):**
 - d-dimensional points in a $[1:10000]^d$ map.
 - 10 (random) anchor points
 - measure generated using a Zipfian distribution; points close to an anchor get higher values.
- **Real spatial data:**
 - Centroids of 400K road segments from North America
 - Measures generated as for synthetic data
- **Comparison includes**
 - aR-tree based top-k OLAP algorithm
 - naive, hash-based method; find the measures for all cells, then select top-k cells. Assumes all cells fit in mem (best case).
- **Default parameters**
 - 200K points; $d=2$ dimensions; $\theta=1$ (Zipf parameter).

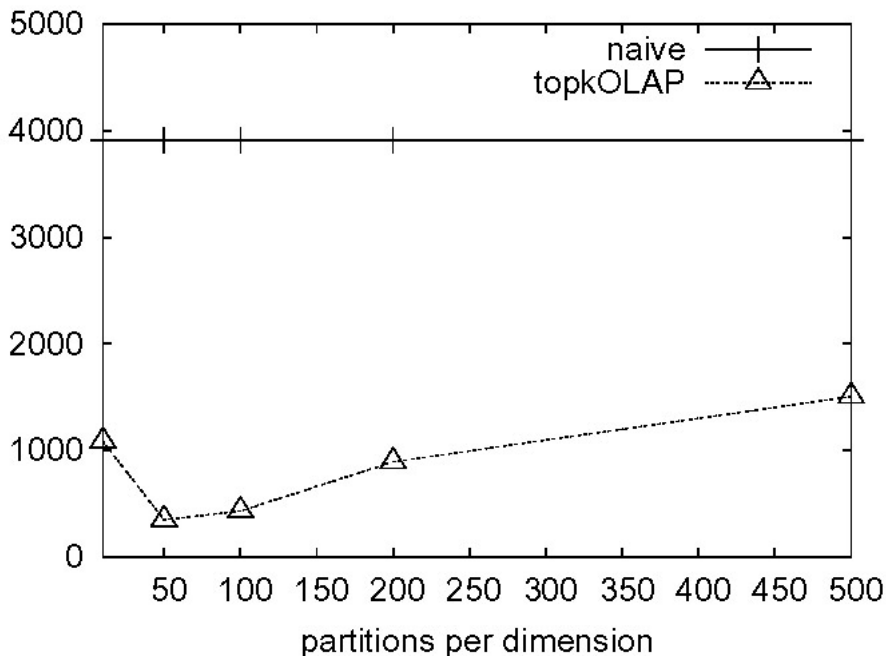
Performance and Scalability (synthetic data)



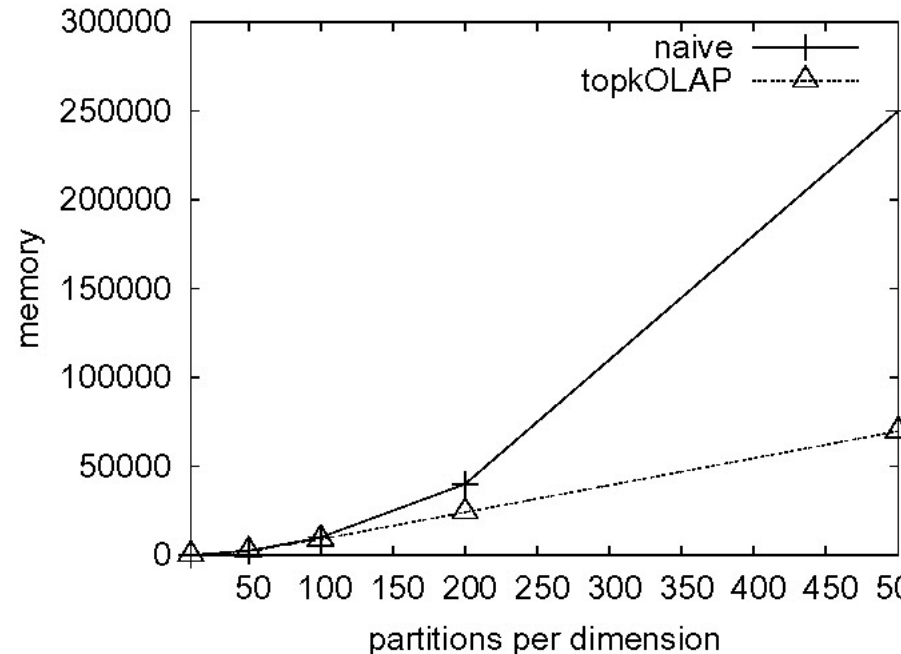
Effect of skew on measures (synthetic data)



Effect of the number of partitions (syn. data)

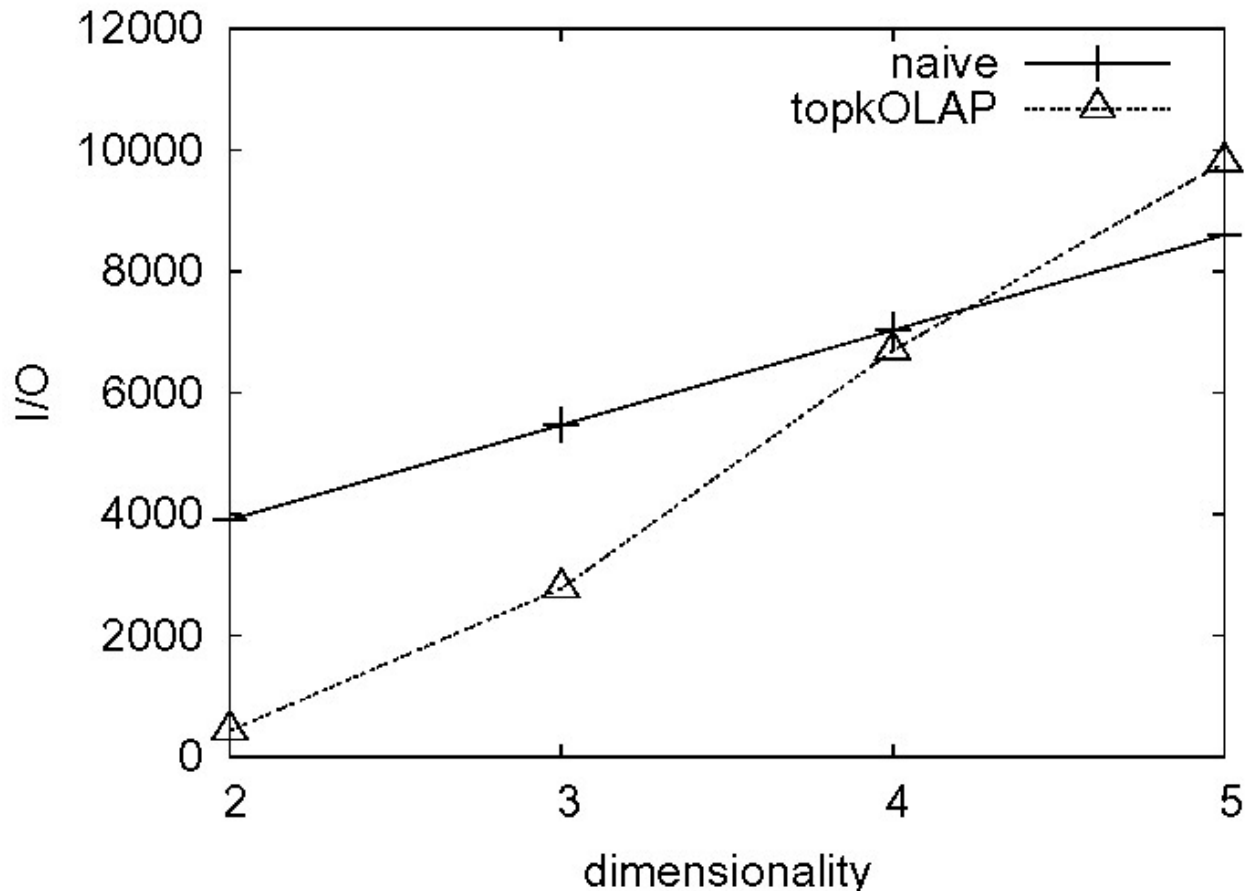


I/O cost

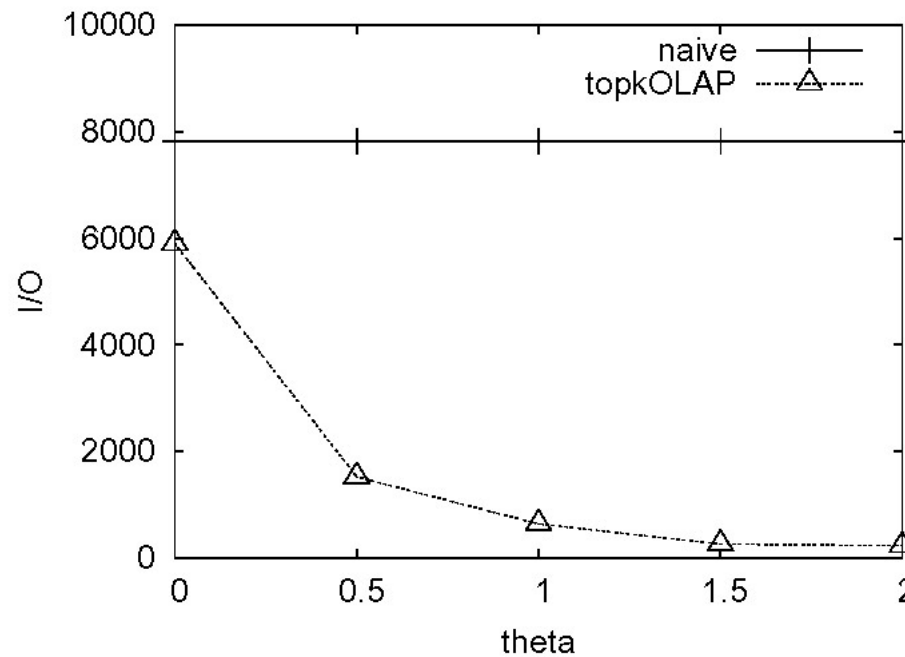
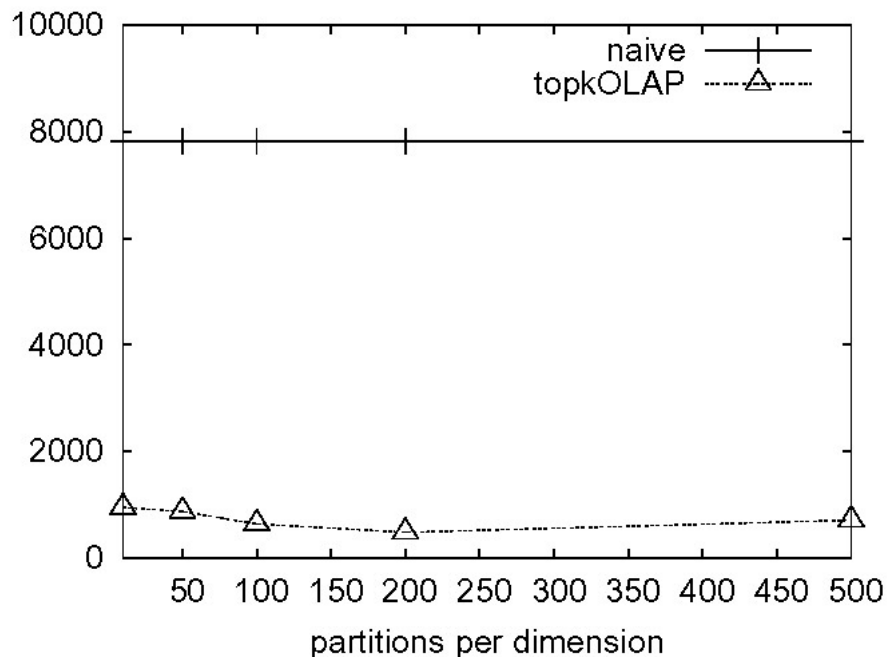


memory requirements

Effect of dimensionality (synthetic data)



Effect of partitions and skew (spatial data)



Conclusions and current work

- ❑ This is the first work (to our knowledge) that studies top-k OLAP queries.
- ❑ We developed an efficient branch-and-bound algorithm for top-k OLAP queries that operates on aR-trees.
- ❑ The algorithm can be easily applied for iceberg queries and other query variants.
- ❑ Experiments show that it performs well for spatial data and low-dimensional data in general.
- ❑ Does not scale well with dimensionality (due to aR-trees and the dimensionality “curse”)
- ❑ Currently working on branch-and-bound algorithms for top-k OLAP queries on non-indexed multi-dimensional data.